**Breadth-First Search (BFS)**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

void bfs(int start, vector<vector<int>>& adj, int n) {

vector<bool> visited(n, false);

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

int node = q.front();

q.pop();

cout << node << " ";

for (int neighbor : adj[node]) {

if (!visited[neighbor]) {

q.push(neighbor);

visited[neighbor] = true;

}

}

}

}

int main() {

int n = 5; // Number of nodes

vector<vector<int>> adj = {

{1, 2},

{0, 3, 4},

{0, 4},

{1},

{1, 2}

};

cout << "BFS Traversal: ";

bfs(0, adj, n);

return 0;

}

**2. Depth-First Search (DFS)**

#include <iostream>

#include <vector>

using namespace std;

void dfs(int node, vector<vector<int>>& adj, vector<bool>& visited) {

visited[node] = true;

cout << node << " ";

for (int neighbor : adj[node]) {

if (!visited[neighbor]) {

dfs(neighbor, adj, visited);

}

}

}

int main() {

int n = 5; // Number of nodes

vector<vector<int>> adj = {

{1, 2},

{0, 3, 4},

{0, 4},

{1},

{1, 2}

};

vector<bool> visited(n, false);

cout << "DFS Traversal: ";

dfs(0, adj, visited);

return 0;

}

**3. Prim's Algorithm**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

void prims(int n, vector<vector<pair<int, int>>>& adj) {

vector<bool> inMST(n, false);

vector<int> key(n, INT\_MAX);

vector<int> parent(n, -1);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;

key[0] = 0;

pq.push({0, 0}); // {weight, node}

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

inMST[u] = true;

for (auto& [v, weight] : adj[u]) {

if (!inMST[v] && weight < key[v]) {

key[v] = weight;

pq.push({key[v], v});

parent[v] = u;

}

}

}

cout << "Edges in MST:\n";

for (int i = 1; i < n; i++) {

cout << parent[i] << " - " << i << endl;

}

}

int main() {

int n = 5;

vector<vector<pair<int, int>>> adj = {

{{1, 2}, {3, 6}},

{{0, 2}, {3, 8}, {4, 5}},

{{4, 7}},

{{0, 6}, {1, 8}},

{{1, 5}, {2, 7}}

};

prims(n, adj);

return 0;

}

**4. Kruskal's Algorithm**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Edge {

int u, v, weight;

bool operator<(Edge const& other) {

return weight < other.weight;

}

};

int find(int v, vector<int>& parent) {

if (v == parent[v]) return v;

return parent[v] = find(parent[v], parent);

}

void unite(int a, int b, vector<int>& parent, vector<int>& rank) {

a = find(a, parent);

b = find(b, parent);

if (a != b) {

if (rank[a] < rank[b]) swap(a, b);

parent[b] = a;

if (rank[a] == rank[b]) rank[a]++;

}

}

void kruskal(int n, vector<Edge>& edges) {

sort(edges.begin(), edges.end());

vector<int> parent(n), rank(n, 0);

for (int i = 0; i < n; i++) parent[i] = i;

vector<Edge> result;

for (Edge& edge : edges) {

if (find(edge.u, parent) != find(edge.v, parent)) {

result.push\_back(edge);

unite(edge.u, edge.v, parent, rank);

}

}

cout << "Edges in MST:\n";

for (Edge& edge : result) {

cout << edge.u << " - " << edge.v << " : " << edge.weight << endl;

}

}

int main() {

int n = 4;

vector<Edge> edges = {

{0, 1, 10}, {0, 2, 6}, {0, 3, 5}, {1, 3, 15}, {2, 3, 4}

};

kruskal(n, edges);

return 0;

}

**5. Dijkstra's Algorithm**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

void dijkstra(int src, int n, vector<vector<pair<int, int>>>& adj) {

vector<int> dist(n, INT\_MAX);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;

dist[src] = 0;

pq.push({0, src});

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

for (auto& [v, weight] : adj[u]) {

if (dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

pq.push({dist[v], v});

}

}

}

cout << "Shortest distances from node " << src << ":\n";

for (int i = 0; i < n; i++) {

cout << i << " : " << dist[i] << endl;

}

}

int main() {

int n = 5;

vector<vector<pair<int, int>>> adj = {

{{1, 10}, {2, 3}},

{{2, 1}, {3, 2}},

{{1, 4}, {3, 8}, {4, 2}},

{{4, 7}},

{{3, 9}}

};

dijkstra(0, n, adj);

return 0;

}